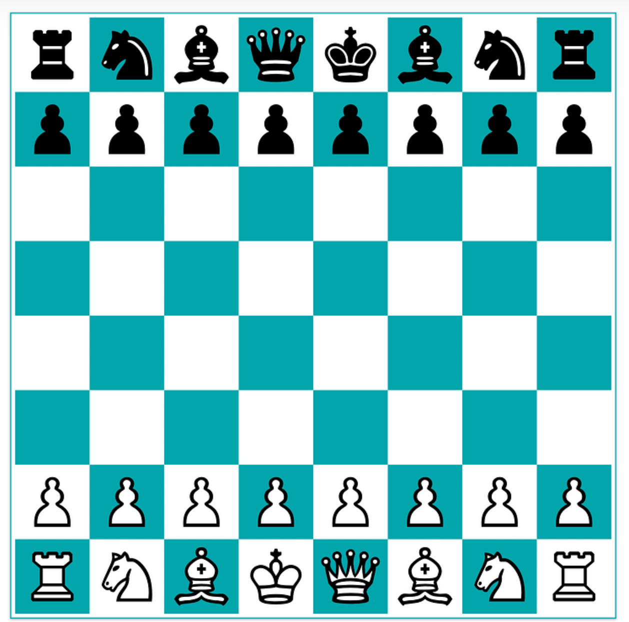
Arrays

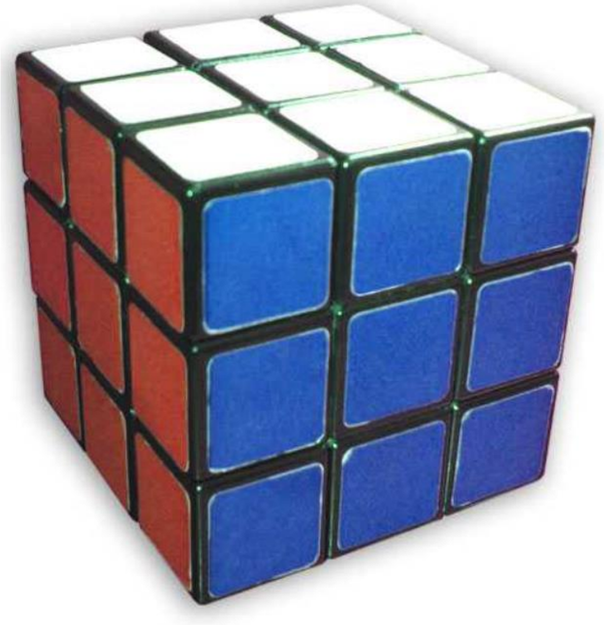
/* Read Chapter 1 of the Course Notes for details. */

Visual Examples of Arrays

https://www.photos-public-domain.com/2011/08/30/truth/

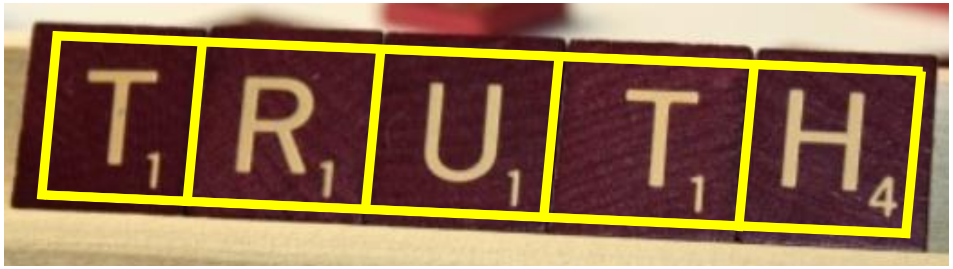


https://pixabay.com/vectors/chessboard-chess-board-game-29630/



https://en.wikipedia.org/wiki/Rubik%27s_Cube#/media/File:Rubiks_cube_solved.jpg

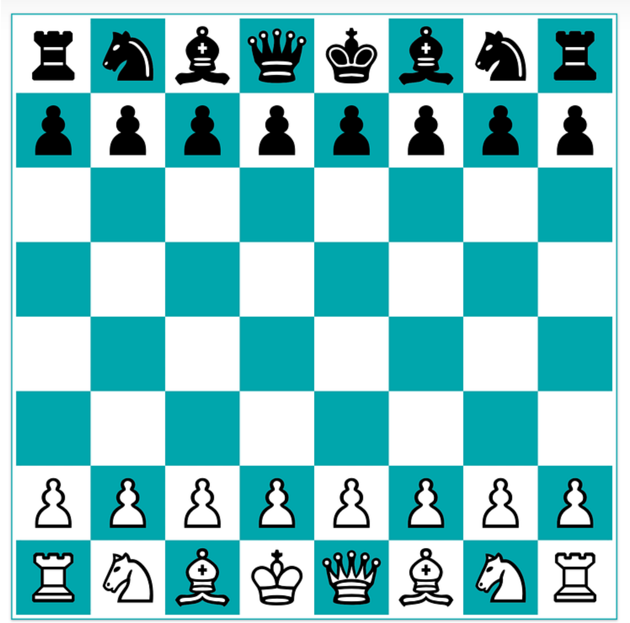
Visual Examples of Arrays



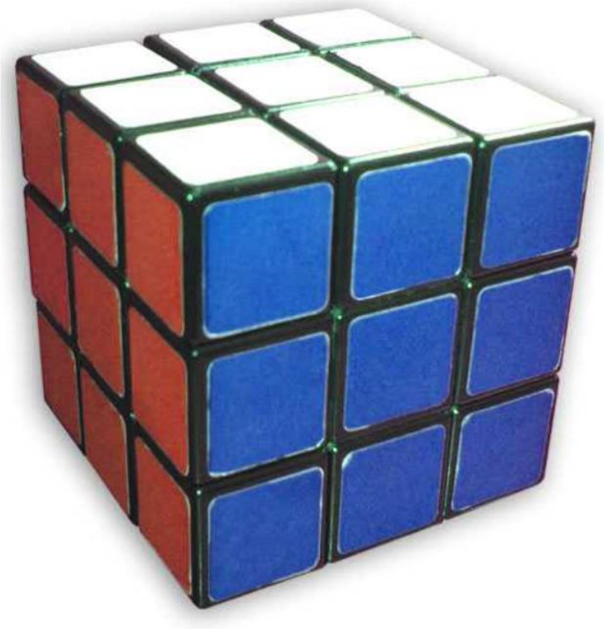
1D array

group of 5 letters

https://www.photos-public-domain.com/2011/08/30/truth/



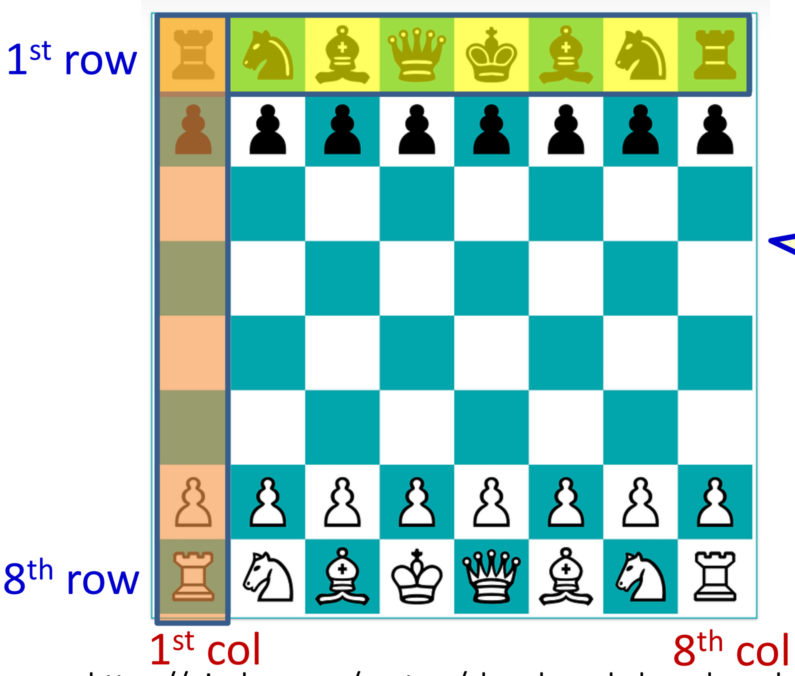
https://pixabay.com/vectors/chessboard-chess-board-game-29630/



https://en.wikipedia.org/wiki/Rubik%27s_Cube#/media/File:Rubiks_cube_solved.jpg

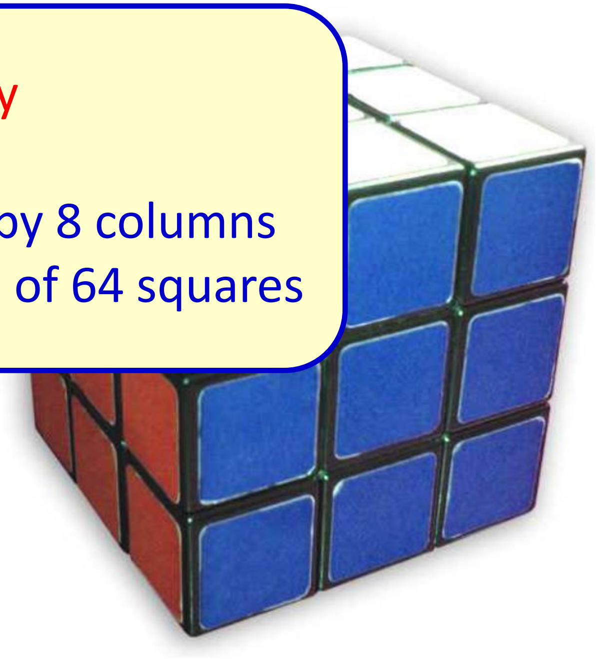
Visual Examples of Arrays

https://www.photos-public-domain.com/2011/08/30/truth/



2D array

8 rows by 8 columns
= group of 64 squares

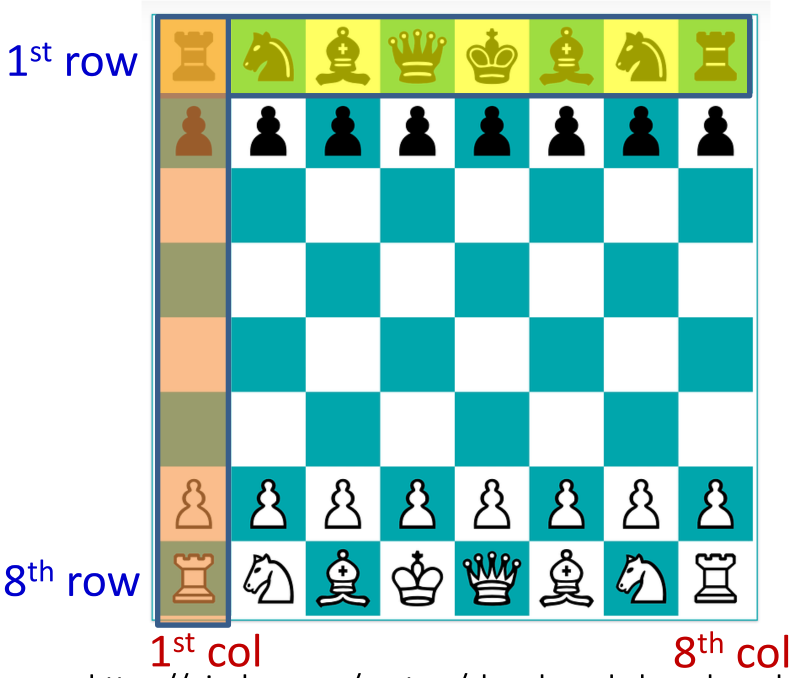


https://pixabay.com/vectors/chessboard-chess-board-game-29630/

https://en.wikipedia.org/wiki/Rubik%27s_Cube#/media/File:Rubiks_cube_solved.jpg

Visual Examples of Arrays

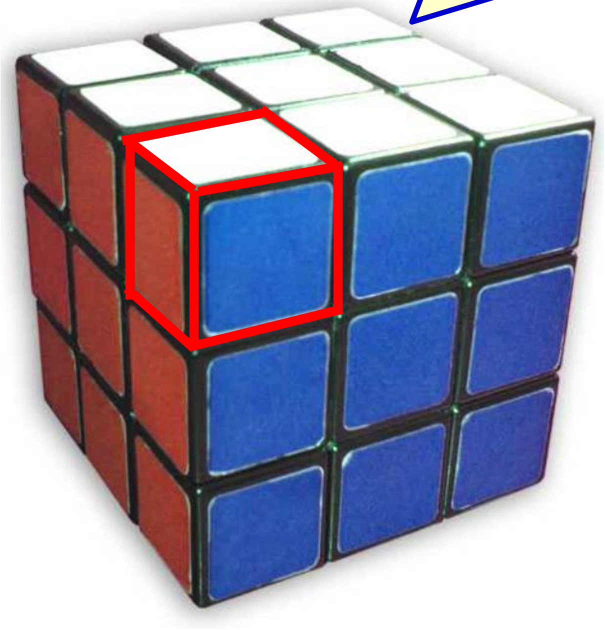
https://www.photos-public-domain.com/2011/08/30/truth/



https://pixabay.com/vectors/chessboard-chess-board-game-29630/

3D array

3 (length) by
3 (width) by
3 (height)
= group of 27 small cubes



https://en.wikipedia.org/wiki/Rubik%27s_Cube#/media/File:Rubiks_cube_solved.jpg

Q: What is an Array?

- is a DATA STRUCTURE variable that stores a group of elements of the data same data type (i.e., homogeneous)
- elements are stored in contiguous memory which is allocated statically
- an array is characterized by:
  - name
  - dimensionality // 1D, 2D, 3D, …
  - size // number of elements
  - element data type

Q: How do you declare an array in C?

- 1D array // also called **“list”**

  <data type> <name> [<size1>]

size must be a non-negative integer

- 2D array // also called **“table”, “matrix”**

  <data type> <name> [<size1>][<size2>]

- 3D array

  <data type> <name> [<size1>][<size2>][<size3>]

Example: Array Declarations

```
// example 1D array declaration
char str[5];
int A[5], B[10];
float Grades[40];
double D[1000];
```

name	:	str
dimensionality	:	1D
size	:	5
element type	:	char

Example: Array Declarations

```
// example 1D array declaration
char str[5];
int A[5], B[10];
float Grades[40];
double D[1000];

// example 2D array declaration
char CrossWord[10][10];   // 10 rows x 10 columns
int Sudoku[9][9];
float Table[2][4];
double Matrix[3][5];
```

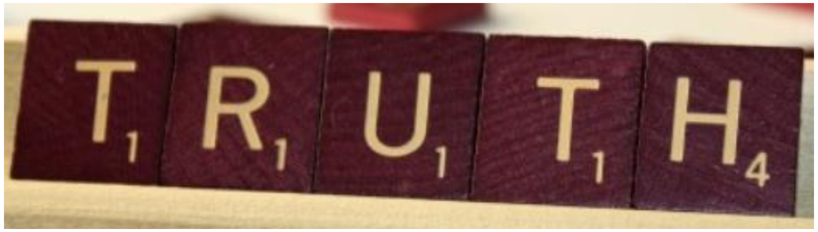
name : Matrix
dimensionality : 2D
size : 3 rows by 5 columns
element type : double

Example: Array Declarations

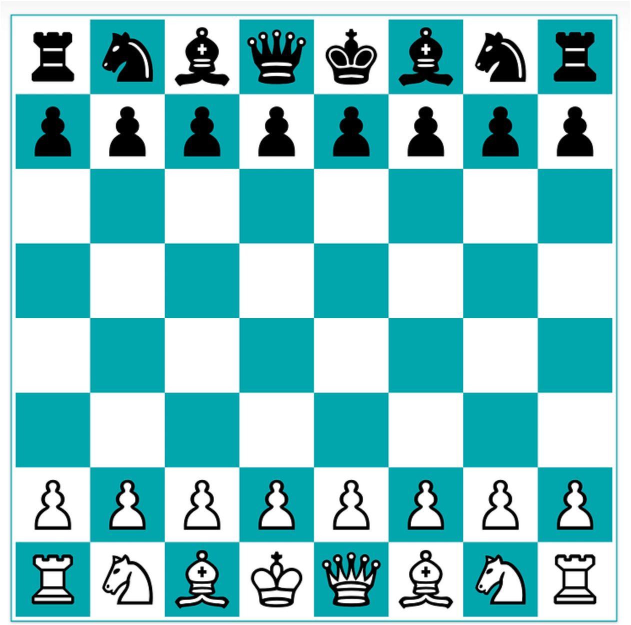
```
// example 1D array declaration
char str[5];
int A[5], B[10];
float Grades[40];
double D[1000];

// example 2D array declaration
char CrossWord[10][10];   // 10 rows x 10 columns
int Sudoku[9][9];
float Table[2][4];
double Matrix[3][5];

// example 3D array declaration
int RubikCube[3][3][3];
```

name	:	RubikCube
dimensionality	:	3D
size	:	3x3x3
element type	:	int

Exercise: Declare array variables for the ff.

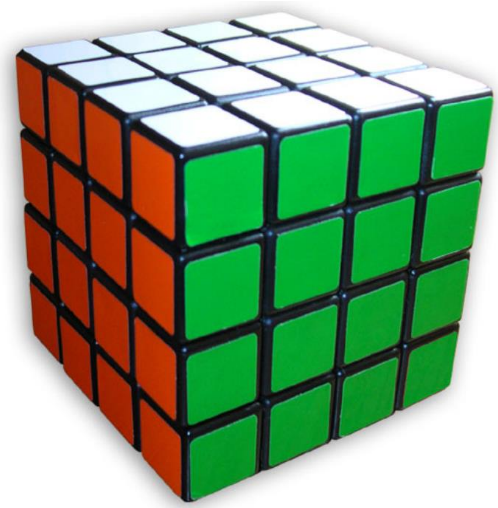


https://www.photos-public-domain.com/2011/08/30/truth/



https://pixabay.com/vectors/chessboard-chess-board-game-29630/

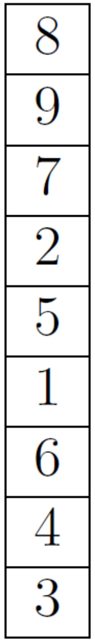
https://www.goodfreephotos.com/other-photos/stack-of-flat-rocks.jpg.php



https://en.wikipedia.org/wiki/Rubik%27s_Revenge#/media/File:Rubiks_revenge_solved.jpg

Let’s concentrate first on 1D arrays.

Q: How do you visualize/draw a 1D Array?



Option 1: draw it vertically

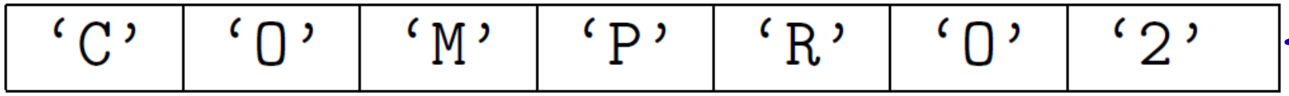
Q: How do you visualize/draw a 1D Array?

8
9
7
2
5
1
6
4
3

Option 2: or draw it horizontally

8	9	7	2	5	1	6	4	3

Q: How do you visualize/draw a 1D Array?



‘C’	‘O’	‘M’	‘P’	‘R’	‘O’	‘2’

Drawing an array of characters

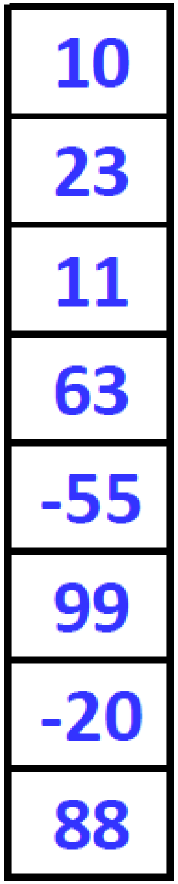
Q: How do you visualize/draw a 1D Array?

‘C’	‘O’	‘M’	‘P’	‘R’	‘O’	‘2’

42.75	-10.23	63.54	89.75	101.23

Drawing an array of floating point values

Q: How do you access an array element?



10

23

11

63

-55

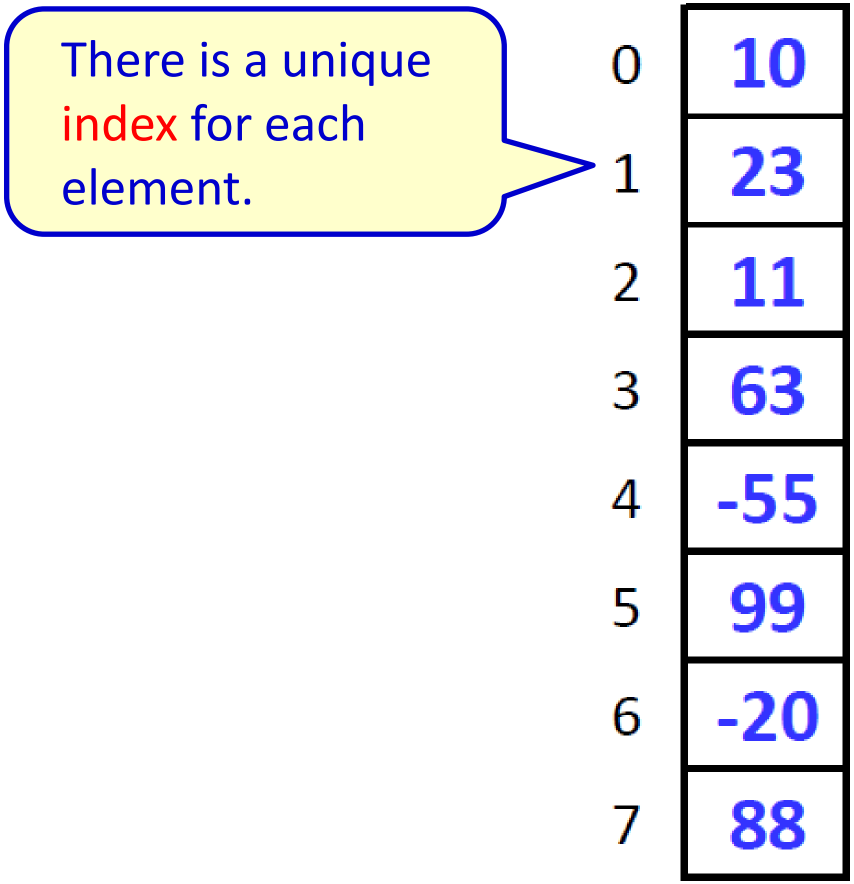
99

-20

88

Declaration: `int A[8];`

Q: How do you access an array element?



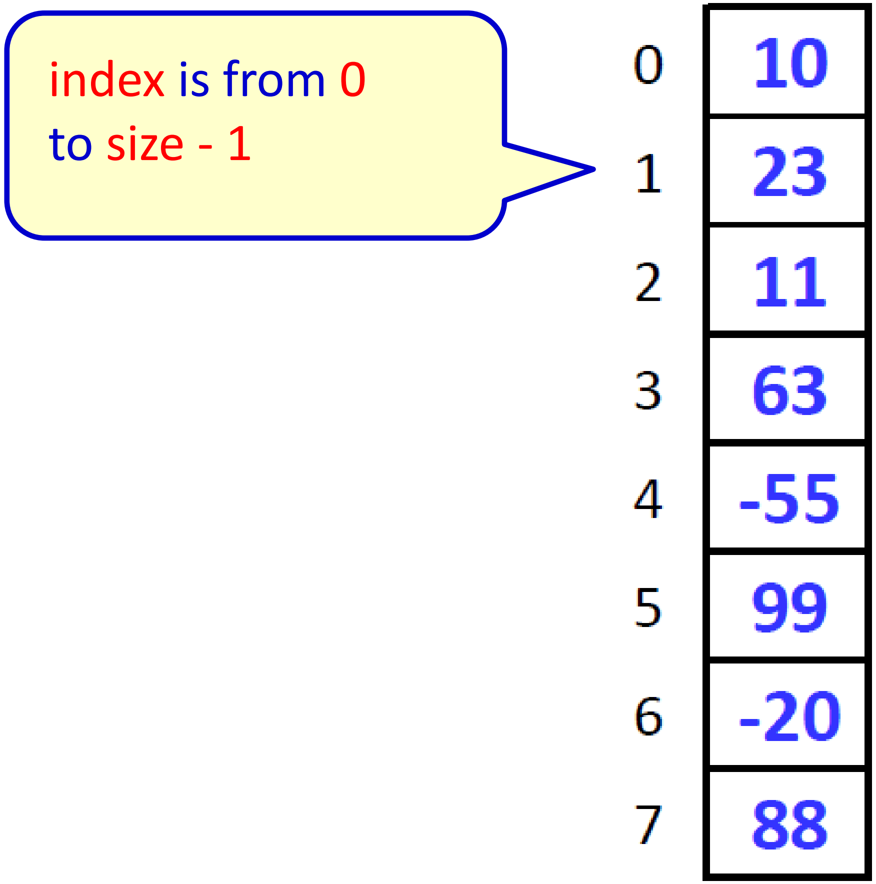
There is a unique index for each element.

0	10
1	23
2	11
3	63
4	-55
5	99
6	-20
7	88

Declaration: `int A[8];`

Q: How do you access an array element?

Declaration: `int A[8];`

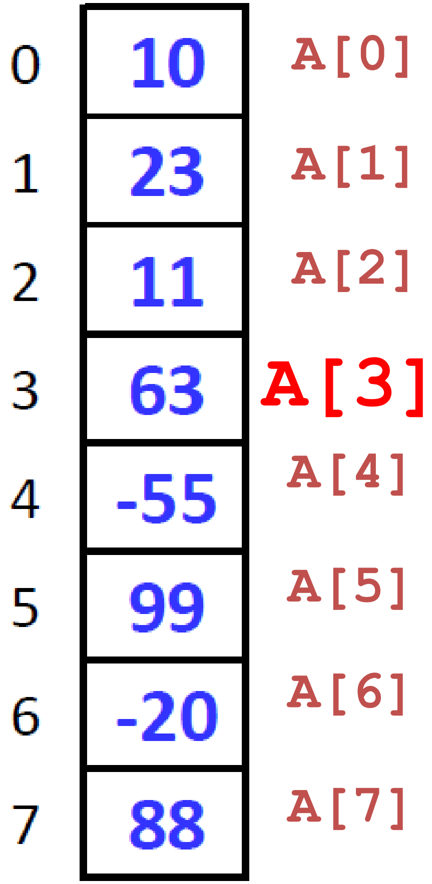


index is from 0 to size - 1

0	10
1	23
2	11
3	63
4	-55
5	99
6	-20
7	88

Q: How do you access an array element?

Declaration: `int A[8];`



0	10	A[0]
1	23	A[1]
2	11	A[2]
3	63	A[3]
4	-55	A[4]
5	99	A[5]
6	-20	A[6]
7	88	A[7]

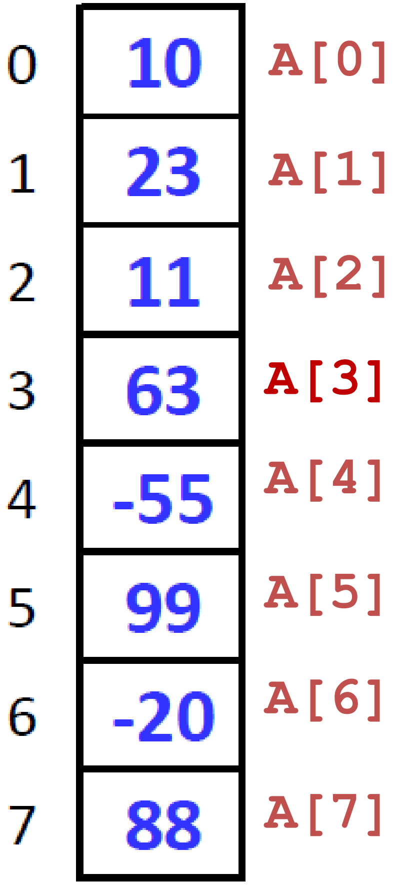
Access an element by giving the array name, and its index enclosed in [ ].

Exercise: Answer the questions below.

Declaration: **`int A[8];`**

What is the value corresponding to the following? If the array access is invalid, explain the reason why.

1. A[0] + A[1]
2. A[-1]
3. A[8]
4. A[8/2]
5. A[8]
6. A[68 – ‘A’]
7. A[A[2]/2]
8. A[-A[6]]
9. A[A[7] % 8]
10. A [A[3] / A[0]]



0	10	A[0]
1	23	A[1]
2	11	A[2]
3	63	**A[3]**
4	-55	A[4]
5	99	A[5]
6	-20	A[6]
7	88	A[7]

Example program

```
#include <stdio.h>
int
main()
{
    char str[5];      // 1D array declaration
    int i;
```

Example program

```
#include <stdio.h>
int
main()
{
    char str[5];      // 1D array declaration
    int i;
```

Questions:

1. What are the values of the elements of array **`str[]`** right after the declaration?
2. How many bytes will be allocated for the use of array **`str[]`**?

Example program

```
#include <stdio.h>
int
main()
{
   char str[5];       // 1D array declaration
   int i;

   str[0] = ‘Z’;      // element initialization
   str[1] = ‘e’;
   str[2] = ‘B’;
   str[3] = ‘$’;
   str[4] = ‘\0’;     // ‘\0’ is the null byte

   for (i = 0; i < 5; i++)
       printf(“%c”, str[i]); // use a loop to access all elements

   return 0;
}
```

Q: How is a 1D array stored in the memory?

A 1D array is stored in in fixed size contiguous memory.

```
// example 1D char array indexing
char str[5];
int i;

str[0] = 'Z';
str[1] = 'e';
str[2] = 'B';
str[3] = '$';
str[4] = '\0';

for (i = 0; i < 5; i++)
   printf("%c", str[i]);
```

Memory Map

	Pointer	Value	
		:	
&str[0]	11AA	Z	str[0]
&str[1]	11AB	e	str[1]
&str[2]	11AC	B	str[2]
&str[3]	11AD	$	str[3]
&str[4]	11AE	\0	str[4]
		:	

Exercise: Answer the questions below.

```
// example 1D int array indexing
int A[5];
int i;

for (i = 0; i < 5; i++)
   scanf("%d", &A[i]);

for (i = 0; i < 5; i++)
   printf("%d", A[i]);
```

	Pointer	Value	
		:	
&A[0]	22BB	8	A[0]
&A[1]	_____	-5	A[1]
&A[2]	_____	9	A[2]
&A[3]	_____	6	A[3]
&A[4]	_____	-23	A[4]
		:	

Questions:

1. What is the data type of A[i]?
2. What is the data type of &A[i]?
3. What is the data type of A?
4. What are the addresses of A[1] to A[4]? Assume sizeof(int) is 4.
5. How many bytes were allocated for the use of array A[]?

Q: Can you pass an array as parameter?

```
void
InputList(int A[], int n)
{
  int i;

  for (i = 0; i < n; i++)
    scanf(“%d”, &A[i]);
}

void
PrintList(int L[], int S)
{
  int j;

  for (j = 0; j < S; j++)
   printf(“%d\n”, L[j]);
}
```

```
int
main()
{
  int A[5];


  InputList(A, 5);

  PrintList(A, 5);

  return 0;
}
```

Questions:

1. What is passed to InputList()?
2. What is passed to PrintList()?

Q: Can you pass an array as parameter?

```
void
InputList(int A[], int n)
{
  int i;

  for (i = 0;
    scanf(“%d”
}

void
PrintList(int L[], int S)
{
  int j;

  for (j = 0; j < S; j++)
   printf(“%d\n”, L[j]);
}
```

No need to indicate size inside [ ]

```
int
main()
{
  int A[5];

       utList(A, 5);

  PrintList(A, 5);

  return 0;
}
```

Questions:

1. What is passed to InputList()?
2. What is passed to PrintList()?

Acdg. to (Kernighan & Ritchie, 1988, page 99)

- “..the name of an array is a synonym for the location of the initial element, …”
- “When an array name is passed to a function, what is passed is the location of the initial element. Within the called function, this argument is a local variable and so an array name parameter is a pointer, that is, a variable containing an address.”
- “In evaluating a[i], C converts it to *(a + i) immediately; the two forms are equivalent.”

&   *   [ ]   relationship

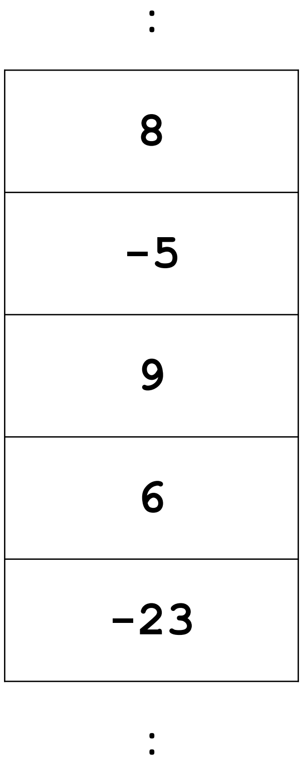
Equivalence 1:

```
A[i] == *(A + i)
```

Equivalence 2:

```
&A[i] == A + i
```

&   *   [ ]   relationship



	Pointer	Value	
		:	
`&A[0] == A + 0`	22BB	8	`A[0] == *(A + 0)`
`&A[1] == A + 1`	22BF	-5	`A[1] == *(A + 1)`
`&A[2] == A + 2`	22C3	9	`A[2] == *(A + 2)`
`&A[3] == A + 3`	22C7	6	`A[3] == *(A + 3)`
`&A[4] == A + 4`	22CB	-23	`A[4] == *(A + 4)`
		:	

Exercise: Fill in the blanks.

```
// array indexing version
int
main()
{
  int i, A[5];

  for (i = 0; i < 5; i++)
      scanf(“%d”, &A[i]);

  for (i = 0; i < 5; i++)
      printf(“%d\n”, A[i]);

  return 0;
}
```

```
// pointer dereferencing version
int
main()
{
  int i, A[5];

  for (i = 0; i < 5; i++)
      scanf(“%d”, __________);

  for (i = 0; i < 5; i++)
      printf(“%d\n”, _________);

  return 0;
}
```

Exercise: Fill in the blanks.

```
// array indexing version
void
InputList(int A[], int n)
{
  int i;

  for (i = 0; i < n; i++)
    scanf(“%d”, &A[i]);
}

void
PrintList(int L[], int S)
{
  int j;

  for (j = 0; j < S; j++)
   printf(“%d\n”, L[j]);
}
```

```
// pointer dereferencing version
void
InputList(__________, int n)
{
  int i;

  for (i = 0; i < n; i++)
    scanf(“%d”, _______);
}

void
PrintList(_______, int S)
{
  int j;

  for (j = 0; j < S; j++)
   printf(“%d\n”, _________);
}
```

Q: How do you define a 1D array?

Syntax:

<data type> <name> [<size>] = { __, __, … }

Example:

```
char str[5] = { ‘Z’, ‘e’, ‘B’, ‘$’, ‘\0’};
int A[5] = { 8, -5, 9, 6, -23 };
int List[10] = { 1, 2, 3 };
```

Q: What are the values of List[3] to List[9]?

Q: What can we do with 1D arrays?

Basic algorithms on a group of items include:

- Minimum - find the lowest value
- Maximum - find the highest value
- Sum - find the total of the values in the group
- Average = sum/n (where n is the # of elements)
- Count - how many values in the group satisfy a given condition?
- Search - Is x in the group?
- Sort - arrange values in increasing/decreasing order

Example: Given the following 1D array, compute the Minimum, Maximum, Sum and Average.

0	10
1	23
2	11
3	63
4	-55
5	99
6	-20
7	88

- Minimum is ___
- Maximum is ___
- Sum is ___
- Average is ___

Example: Given the following 1D array, answer the following counting problems?

0	10
1	23
2	11
3	63
4	-55
5	99
6	-20
7	88

How many are elements are

- positive? ___
- negative? ___
- even? ___
- odd? ___
- higher than 50? ___
- there between -10 and 50 inclusive? ____

Minimum Algorithm

```
#define SIZE 8

// Find the smallest value in the group.  Assume that the values are unique.
// Return the index of the minimum element.
int
Minimum(int A[], int n)
{
  int i;
  int min = 0; // assume that 1st element is the smallest

  for (i = 1; i < n; i++) // note: start with index 1 not 0
      if (A[min] > A[i])
          min = i; // update the minimum index

  return min;
}

int
main()
{
    int A[SIZE] = {10, 23, 11, 63, -55, 99, -20, 88};

    printf(“%d\n”, Minimum(A, SIZE));
    return 0;
}
```

Exercise: Maximum Algorithm

```
// Find the highest value in the group.  Assume that
// the values are unique.
// Return the index of the maximum element.
int
Maximum(int A[], int n)
{
  int i;



  /* Implement the body of this function. */



}
```

Sum Algorithm

```
// Find the sum of the values in the group.
// Return the sum (also called total).
int
Sum(int A[], int n)
{
  int i;
  int acc = 0; // don't forget to initialize the accumulator

  for (i = 0; i < n; i++)
      acc = acc + A[i];  // acc += A[i];

  return acc;
}
```

Exercise: Average Algorithm

```
// Find the average of the values in the group.
// Return the average.
float
Average(int A[], int n)
{
  int i;

    /* Implement the body of this function. */

}
```

Counting Positive Numbers Algorithm

```
// Count the number of positive elements in the group.
// Return the count.
int
CountPositive(int A[], int n)
{
  int i;
  int ctr = 0; // don’t forget to initialize the ctr

  for (i = 0; i < n; i++)
      if (A[i] > 0)  // check the necessary condition
          ctr++;

  return ctr;
}
```

Exercise: Count the Odd Number

```
// Count the number of odd numbers in the group.
// Return the count.
int
CountOdd(int A[], int n)
{
  int i;

    /* Implement the body of this function. */

}
```

Search Algorithm

Problem: Determine if a given search key value is (found) in a known universe of key values. Use a 1D array with size *n* to represent the list of unique keys.

Solution:

1. Linear search algorithm: *O(n)*

   Watch: https://www.youtube.com/watch?v=TwsgCHYmbbA

2. Binary search algorithm: *O(lg n)* but requires that the array be in sorted order

   Watch: https://www.youtube.com/watch?v=T98PIp4omUA

Linear Search Algorithm

```
// Search for key in A[] with n elements.
// Return index if found, otherwise, return -1
int
LinearSearch(int key, int A[], int n)
{
  int i;

  for (i = 0; i < n; i++)
      if (key == A[i]) // found
         return i;

  return -1; // not found if we reach this point
}
```

Questions:

1. How many times, at the minimum, will the if () be performed?
2. How many times, at the maximum, will the if() be performed?

Example: Determine the value returned by the LinearSearch() function call based on the following 1D array.

0	10
1	23
2	11
3	63
4	-55
5	99
6	-20
7	88

- LinearSearch(10, A, 8) is ____.
- LinearSearch(-55, A, 8) is ____.
- LinearSearch(22, A, 8) is ____.

Binary Search Algorithm

```
int
BinarySearch(int key, int A[], int n)
{
  int low = 0, high = n - 1, mid;
  int found = 0;

  while (!found && low <= high) {
     mid = low + (high – low)/2;
     if (key == A[mid]) // found
          found = 1;
     else if (key < A[mid]) // search lower half
              high = mid – 1;
          else // key > A[mid]  search upper half
              low = mid + 1; // search upper half
  }

  if (found)
       return mid;
  else
      return -1;
}
```

Sorting Algorithm

Problem: Given an unordered set of key values, re-arrange them such that they will be in increasing order (or decreasing order).

Use a 1D array with size *n* to represent the list of key values. The array is sorted in increasing order when **`A[i] <= A[i+i]`** for *i = 0* to *n - 2*.

Solution: Selection sort (discussed in CCPROG2 Course Notes)

Watch: https://www.youtube.com/watch?v=3hH8kTHFw2A

Selection sort

```
void
SelectionSort(int A[], int n)
{
  int i, j, min, temp;

  for (i = 0; i < n - 1; i++) {
      min = i;  // min is the index of the lowest element

      for (j = i + 1; j < n; j++)
          if (A[min] > A[j])
             min = j;

      // swap A[i] with A[min]
      if (i != min) {
        temp = A[i];
        A[i] = A[min];
        A[min] = temp;
      }
  }
}
```

Selection sort with a separate Swap() function

```
void
Swap(int *px, int *py)
{
  int temp;

   temp = *px;
   *px = *py;
   *py = temp;
}
```

```
void
SelectionSort(int A[], int n)
{
  int i, j, min;

  for (i = 0; i < n - 1; i++) {
      min = i;

      for (j = i + 1; j < n; j++)
          if (A[min] > A[j])
             min = j;

     // swap A[i] with A[min]
     if (i != min)
         Swap(&A[i], &A[min]);
   }
}
```

Q: Are there other sorting algorithms?

Yes -- you will learn them in your future subjects on Data Structures & Algorithms, and Algorithm Complexity.

Refer to: https://visualgo.net/en/sorting

Mergesort: https://www.youtube.com/watch?v=Ns7tGNbtvV4

Quicksort: https://www.youtube.com/watch?v=ywWBy6J5gz8

Q: Can you manipulate more than 1 array?

```
void
Copy(int destination[], int source[], int n)
{
  int i;

  for (i = 0; i < n; i++)
       destination[i] = source[i];
}

int
main()
{
  int A[5] = {10, 20, 30, 40, 50};
  int B[5];

  Copy(B, A, 5);
  // other codes here…
  return 0;
}
```

Q1: What are the contents of array B before calling Copy()?

Q2: What are the contents of array B after calling Copy()?

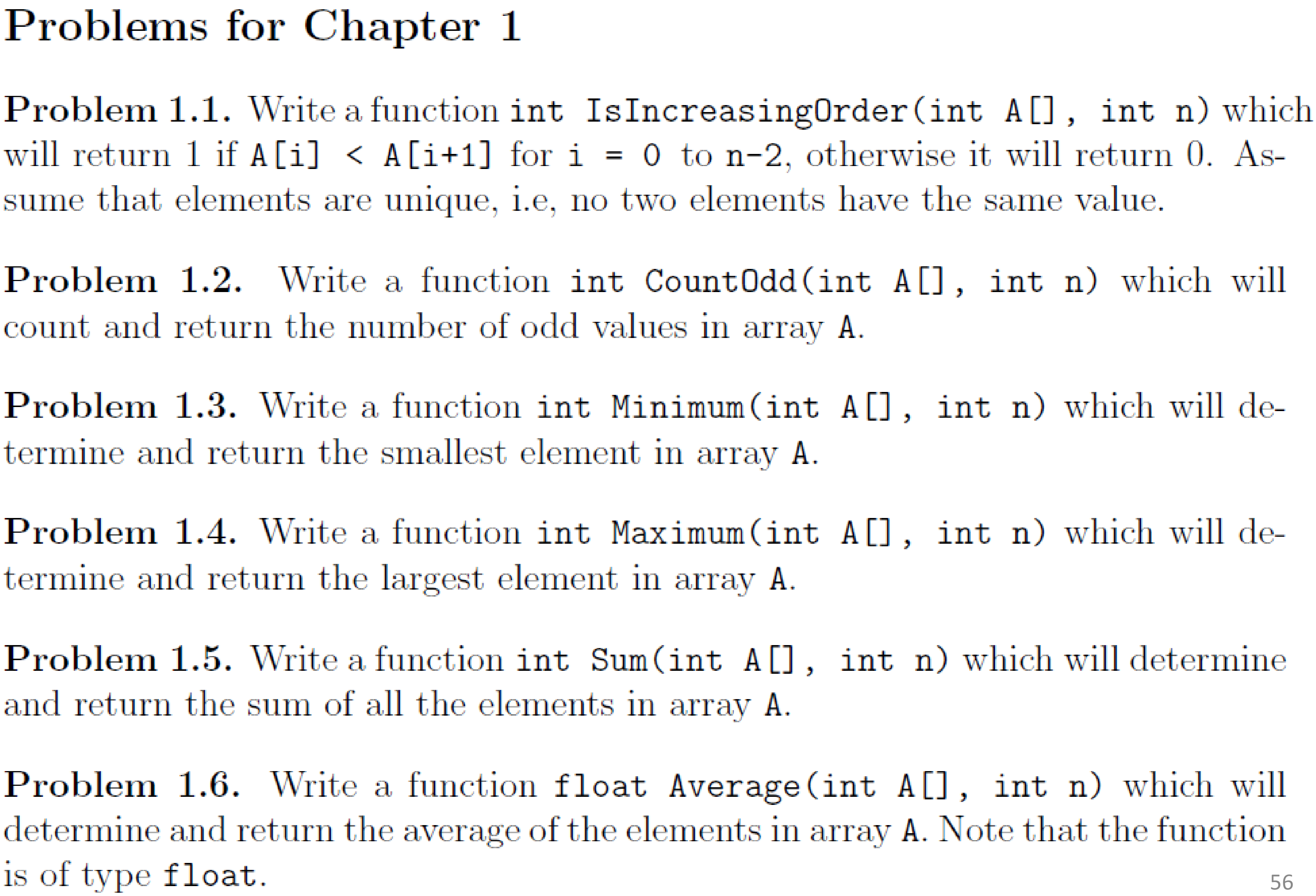
Q: Can you manipulate more than 1 array?

```
void
VectorAdd(int A[], int B[], int C[], int n)
{
  int i;

  for (i = 0; i < n; i++)
       C[i] = A[i] + B[i];
}

int
main()
{
  int A[5] = {10, 20, 30, 40, 50}
  int B[5] = {-20, 25, -2, 100, -90};
  int C[5];

  VectorAdd(A, B, C, 5);
  // other codes here…
  return 0;
}
```

Q1. What are the contents of array C before calling VectorAdd()?

Q2. What are the contents of array C after calling VectorAdd()?

Exercise: Solve the following problems (from our Course Notes)



Problems for Chapter 1

Problem 1.1. Write a function `int IsIncreasingOrder(int A[], int n)` which will return 1 if `A[i] < A[i+1]` for `i = 0` to `n-2`, otherwise it will return 0. Assume that elements are unique, i.e, no two elements have the same value.

Problem 1.2. Write a function `int CountOdd(int A[], int n)` which will count and return the number of odd values in array `A`.

Problem 1.3. Write a function `int Minimum(int A[], int n)` which will determine and return the smallest element in array `A`.

Problem 1.4. Write a function `int Maximum(int A[], int n)` which will determine and return the largest element in array `A`.

Problem 1.5. Write a function `int Sum(int A[], int n)` which will determine and return the sum of all the elements in array `A`.

Problem 1.6. Write a function `float Average(int A[], int n)` which will determine and return the average of the elements in array `A`. Note that the function is of type `float`.

Problem 1.7 Write a function `int CountUpper(char S[], int n)` which will count the number of upper case letters in array `S`. For example, assume an array `S` defined as follows: `char S[7] = {’C’, ’o’, ’M’, ’p’, ’r’, ’o’, ’2’};` A call to `CountUpperCase(S, 7)` will return 2 since there are two upper case letters, namely, ’C’ and ’M’.

Problem 1.8 Write a function `void ConvertUpper(char S[], int n)` which will convert all letters in the array to upper case. For example, `ConverUpper(S, 7)` using the array `S` defined in the previous problem will result into a modified array `S` containing ’C’, ’O’, ’M’, ’P’, ’R’, ’O’, ’2’.

Problem 1.9. Write a function `void MaxCopy(int C[], int A[], int B[], int n)`. Assume that arrays `A` and `B` contain values. Determine which of the two values between `A[i]` and `B[i]` is higher. The higher value is assigned to `C[i]` for `i = 0` to `n-1`.

Next, we cover 2D arrays.

Q: How do you access 2D array elements?

```
// example 2D char array declaration
char C[2][3];  // 2D array declaration

// row 0 elements
C[0][0] = ‘C’;
C[0][1] = ‘O’;
C[0][2] = ‘M’;
// row 1 elements
C[1][0] = ‘P’;
C[1][1] = ‘R’;
C[1][2] = ‘O’;
// print elements in row major order
for (row = 0; row < 2; row++) {
    for (col = 0; col < 3; col++)
        printf(“%c “, C[row][col]);
    printf(“\n”);
}
```

	col 0	col 1	col 2
row 0	C	O	M
row 1	P	R	O

Q: How is a 2D array stored in the memory?

	col 0	col 1	col 2
row 0	C	O	M
row 1	P	R	O

A 2D array is stored in row major order (fixed size contiguous memory).

&C[0][0]	11AA	C	C[0][0]
&C[0][1]	11AB	O	C[0][1]
&C[0][2]	11AC	M	C[0][2]
&C[1][1]	11AD	P	C[1][1]
&C[1][2]	11AE	R	C[1][2]
&C[1][2]	11AF	O	C[1][2]

Example program

```
#include <stdio.h>
#define NROWS 3
#define NCOLS 2
int
main()
{
   int T[NROWS][NCOLS];  // 2D array declaration
   int i, j;

   for (i = 0; i < NROWS; i++)  // array initialization
     for (j = 0; j < NCOLS; j++)
       scanf(“%d”, &T[i][j]);

   for (i = 0; i < NROWS; i++)  // print array elements
      for (j = 0; j < NCOLS; j++)
         printf(“%d\n”, T[i][j]);

   return 0;
}
```

Example program

```
#define NROWS 3
#define NCOLS 2
void
InitializeMatrix(int A[][NCOLS])
{
   int i, j;

   for (i = 0; i < NROWS; i++)
      for (j = 0; j < NCOLS; j++)
         scanf(“%d”, &A[i][j]);
}

void
PrintMatrix (int B[][NCOLS])
{
   int i, j;

   for (i = 0; i < NROWS; i++) {
      for (j = 0; j < NCOLS; j++)
           printf(“%d ”, B[i][j]);

      printf(“\n”);
   }
}
```

```
int
main()
{
   int T[NROWS][NCOLS]; // 2D array declaration

   InitializeMatrix(T);

   PrintMatrix(T);

   return 0;
}
```

Column size is necessary. But there is NO need to indicate the row size.

Exercise: Based on the array **M[][]** definition below, write the addresses and values in the memory map.

```
double M[3][2] = { {6.6, 5.5}, {4.4, 3.3}, {2.2, 1.1} };
```

	Address	Value	
		:	
&M[0][0]	88AA	____	M[0][0]
&M[0][1]	____	____	M[0][1]
&M[1][0]	____	____	M[1][0]
&M[1][1]	____	____	M[1][1]
&M[2][0]	____	____	M[2][0]
&M[2][1]	____	____	M[2][1]

Exercise: Based on the array `M[][]` definition in the previous slide, implement `void PrintRowMajor(double M[3][2])` which will produce the following one-line output on the screen:

```
6.6  5.5  4.4  3.3  2.2  1.1
```

```
void PrintRowMajor(double M[3][2])
{
    // use nested loop(s), brute force NOT allowed




}
```

Exercise: Based on the array **M[][]** definition in the previous slide, implement **void PrintColumnMajor(double M[3][2])** which will produce the following one-line output on the scren:

```
6.6   4.4   2.2   5.5   3.3   1.1
```

```
void PrintColumnMajor(double M[3][2])
{
    // use nested loop(s), brute force NOT allowed



}
```

Q: What can we do with 2D arrays?

Basic algorithms on a group of items include:

- Minimum - find the lowest value
- Maximum - find the highest value
- Sum - find the total of the values in the group
- Average = sum/n (where n is the # of elements)
- Count - how many values in the group satisfy a given condition?
- Search - Is x in the group?
- Sort - arrange values in increasing/decreasing order

Exercise: Given the following 2D array, compute the Minimum, Maximum, Sum and Average.

Assume the following codes

```
#define NROWS 3
#define NCOLS 5

int M[NROWS][NCOLS];
```

Assume also that the contents of M are:

	col 0	col 1	col 2	col 3	col 4
row 0	**-5**	**2**	**1**	**4**	**9**
row 1	**6**	**8**	**22**	**-7**	**8**
row 2	**3**	**15**	**-5**	**10**	**25**

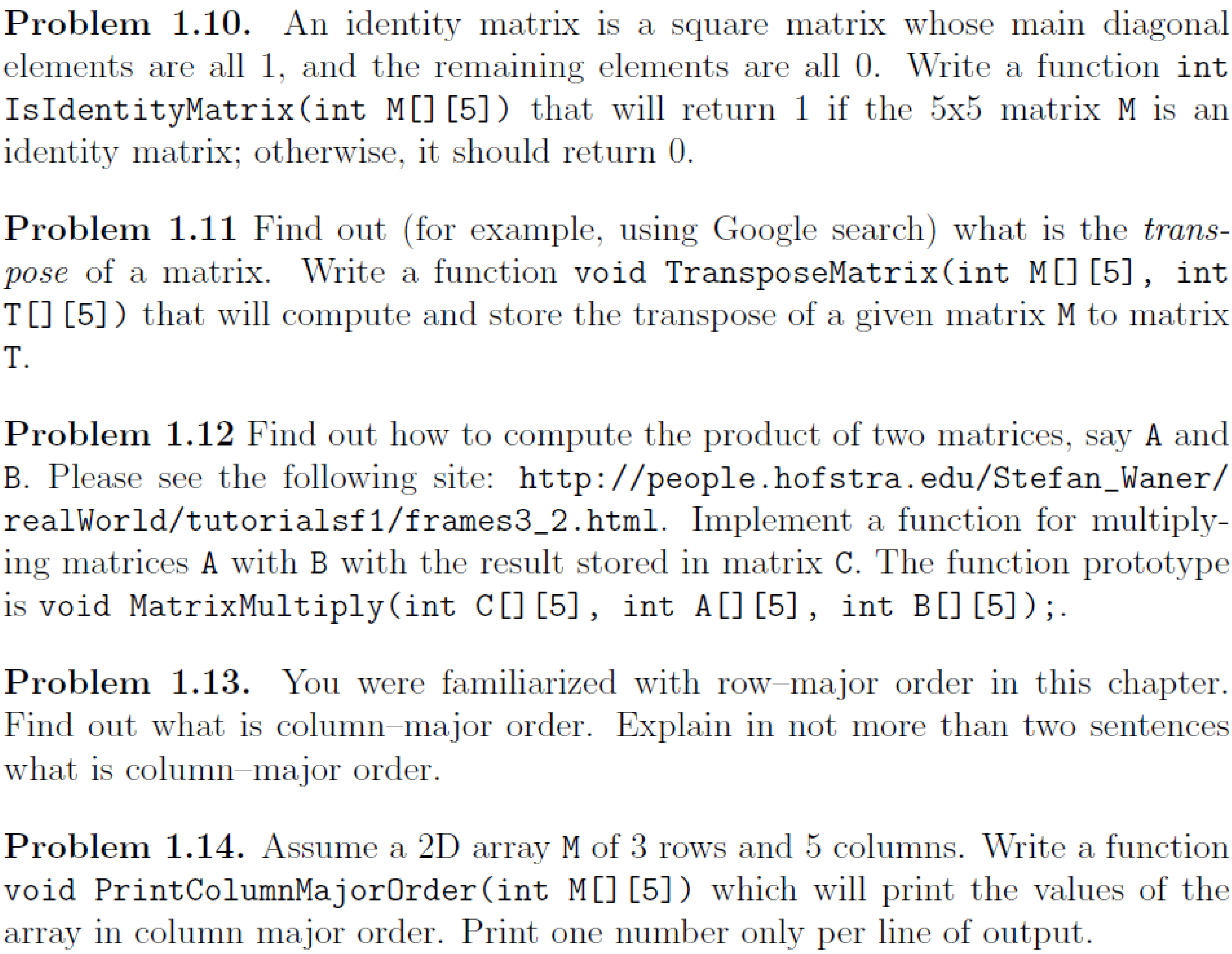
- Minimum is ___
- Maximum is ___
- Sum is ___
- Average is ___
- Row 1 Sum is ___
- Col 3 Sum is ___

Exercise:

Assume a 2D array int M[NROWS][NCOLS]. Implement the ff functions:

1. int Minimum(int M[][NCOLS]) – return the smallest element in M.
2. int Sum(int M[][NCOLS]) – return the sum of the elements in M.
3. int RowSum(int M[][COLS], int row_index) – return the sum of the elements in row row_index.
4. int ColSum(int M[][COLS], int row_index) – return the sum of the elements in column col_index.
5. float Average(int M[][NCOLS]) – return the average of the elements in M.
6. int * Search(int M[][NCOLS], int key) – do a linear search on M in row major order. If the key is found, return its memory address; otherwise return NULL.

Exercise: Solve the following problems (from our Course Notes)



Problem 1.10. An identity matrix is a square matrix whose main diagonal elements are all 1, and the remaining elements are all 0. Write a function int IsIdentityMatrix(int M[][5]) that will return 1 if the 5x5 matrix M is an identity matrix; otherwise, it should return 0.

Problem 1.11 Find out (for example, using Google search) what is the *transpose* of a matrix. Write a function void TransposeMatrix(int M[][5], int T[][5]) that will compute and store the transpose of a given matrix M to matrix T.

Problem 1.12 Find out how to compute the product of two matrices, say A and B. Please see the following site: http://people.hofstra.edu/Stefan_Waner/realWorld/tutorialsf1/frames3_2.html. Implement a function for multiplying matrices A with B with the result stored in matrix C. The function prototype is void MatrixMultiply(int C[][5], int A[][5], int B[][5]);.

Problem 1.13. You were familiarized with row–major order in this chapter. Find out what is column–major order. Explain in not more than two sentences what is column–major order.

Problem 1.14. Assume a 2D array M of 3 rows and 5 columns. Write a function void PrintColumnMajorOrder(int M[][5]) which will print the values of the array in column major order. Print one number only per line of output.

The ideas discussed so far can be extended to arrays of higher dimension than 2D (i.e., 3D, 4D…).

-- The End --